Institut für Physikalische Chemie

Methodenkurs „Anwendungen von Mathematica und Matlab in der Physikalischen Chemie“ im Sommersemester 2016

Prof. Dr. Stefan Weber, Dr. Till Biskup

— Lösungen zum Aufgabenblatt 4 zum Teil „Matlab“ vom 26.07.2016 —

Vorbemerkung

Die nachfolgend vorgestellten Lösungen wurden (unter MATLAB 2014a) getestet und sollten funktionieren. In den meisten Fällen gibt es aber mehr als eine Lösung, und die hier vorgestellte ist mit großer Wahrscheinlichkeit nicht die eleganteste oder beste und auf keinen Fall die „allein glückselig machende“.

Grundsätzlich gilt: Versuchen Sie immer, den Quellcode anderer Menschen in jedem Detail zu verstehen, bevor Sie ihn einsetzen. Nur so lernen Sie für sich selbst etwas dazu.

Aufgabe 4—1 (Quellcodebereinigung)

In Listing 1 finden Sie den von Fehlern und Warnungen bereinigten Quellcode. Vergleichen Sie ihn mit dem ursprünglichen Quellcode von der Internetseite[1] des Methodenkurses.

Ein paar Anmerkungen zum Quellcode in seiner bereinigten Form:

- Wie Sie sehen, wurde die Einrückung automatisch vereinheitlicht. Das erhöht insbesondere für Schleifen die Lesbarkeit des Quellcodes. Sie finden diese Funktion im MATLAB-Editor im Menü „Text“ → „Smart Indent“.
- Einen Aspekt, den der MATLAB-Editor nicht oder nur in seltenen Fällen bemängelt, sind mehrere durch Semikolon getrennte Definitionen oder Befehle in einer Zeile. Trotzdem erhöht es immens die Übersichtlichkeit, wenn Sie hier die entsprechenden Definitionen jeweils in einer eigenen Zeile vornehmen.
- Noch wurden keine Kommentare oder Leerzeichen zwischen einzelnen zusammengehörigen Codeblöcken eingeführt. Das Ergebnis dieses logisch nächsten und wichtigen Schrittes wird in der folgenden Aufgabe dokumentiert.

Eine Besonderheit findet sich in Zeile 4 von Listing 1: Die Angabe von „`%#ok<SAGROW>`“ teilt dem MATLAB-Editor[2] mit, in dieser Zeile eine anderenfalls angezeigte Warnung zu unterdrücken. Hier ändert ein Array innerhalb einer Schleife seine Größe, was die Geschwindigkeit der Ausführung stark verringert. Allerdings lässt sich hier ein derartiges Konstrukt schwer verhindern, da die Größe der einzulesenden Datei im Vorfeld ja nicht bekannt ist. Eine andere Lösung, den MATLAB-Code-Analysator zufrieden zu stellen, sehen Sie in der zweiten Aufgabe in Listing 2.

[1] `http://www.till-biskup.de/de/lehre/mathematica-matlab/ss2016/material/06/index`

[2] Strenggenommen ist das ein Hinweis an den statischen Code-Analysator von MATLAB, mlint.

Ein Wort der Vorsicht: Natürlich lassen sich viele Warnungen im MATLAB-Editor auf diese Weise unterdrücken, zumal der Editor Ihnen nicht nur die Möglichkeit bietet, die Warnung auf einer Zeile zu unterdrücken, sondern auch global in der gesamten Datei. Das ist in der Regel aber eher eine schlechte Idee und sollte *niemals* das saubere Programmieren ersetzen. Grundsätzlich gilt: Nehmen Sie Warnungen ernst, die Ihnen eine statische Code-Analyse gibt, und beheben Sie die Ursache(n). Guter Code zeichnet sich nicht nur durch Fehlerfreiheit, sondern durch Eleganz und Sauberkeit in den Details aus.

Listing 1: Fehlerbereinigter Quellcode

```
fid=fopen('uvvis.txt');
k=1;
while ~feof(fid)
    line{k}=fgetl(fid); %#ok<SAGROW>
    k=k+1;
end
fclose(fid);
line(1:2)=[];
data=zeros(length(line),2);
for k=1:length(line)
    data(k,:)=cell2mat(textscan(strrep(line{k},',','.'),'%f %f'));
end
bg=load('bg.txt');
bg=bg(1:length(data),:);
bg(:,2)=bg(:,2)-bg(end,2);
bg(:,2)=bg(:,2)*(data(data(:,1)==520,2)/bg(bg(:,1)==520,2));
data(:,2)=data(:,2)-bg(:,2);
plot(data(:,1),data(:,2),'k-');
hold on;
plot(data(:,1),zeros(length(data(:,2)),1),'k--');
hold off;
xlabel('{\it wavelength} / nm');
ylabel('{\it intensity} / OD');
set(get(gca,'xlabel'),'fontsize',12);
set(get(gca,'ylabel'),'fontsize',12);
set(get(gca,'xlabel'),'fontname','Arial');
set(get(gca,'ylabel'),'fontname','Arial');
set(gca,'fontsize',12);
set(gca,'fontname','Arial');
set(gcf,'paperunits','centimeters');
set(gcf,'papersize',[16 10]);
set(gcf,'paperpositionmode','auto');
set(gca,'Units','centimeters');
set(gca,'OuterPosition',[0 0 16 10]);
set(gcf,'Units','centimeters');
oldpos = get(gcf,'Position');
set(gcf,'Position',[oldpos([1 2]) 16 10]);
print(gcf,'data.pdf','-dpdf');
```

Aufgabe 4—2 (Quellcode gliedern und dokumentieren)

Das tiefe Verständnis des Quellcodes in allen Details kann letztlich nur in Eigenregie erarbeitet werden. Machen Sie sich selbst die Mühe und nutzen Sie bei eventuellen Unklarheiten oder Unsicherheiten die Ihnen zur Verfügung stehenden Quellen (insbesondere die eingebaute Hilfe von MATLAB).

Ziel des Skriptes ist das Einlesen eines Absorptionsspektrums, die Darstellung der Daten und die anschließende Speicherung in einem außerhalb von MATLAB verwendbaren Format (hier: PDF). Die einzelnen Funktionen, die das Skript ausführt, lassen sich grob wie folgt zusammenfassen:

a) Einlesen der eigentlichen Messdaten,

b) Hintergrund laden und abziehen,

c) Daten grafisch darstellen,

d) Abbildung (als PDF-Dokument) speichern.

Manche dieser einzelnen Aufgaben lassen sich weiter unterteilen. Im nachfolgend abgedruckten Listing 2 finden Sie eine Möglichkeit, den Ausgangsquellcode in Blöcke zu unterteilen und die einzelnen Bereiche mit kurzen Kommentaren zu dokumentieren.

Ein paar Anmerkungen zum Quellcode in seiner in Blöcke unterteilten und grob dokumentierten Form:

- Wie Sie sehen können, handelt es sich bei den eigentlichen Messdaten und beim Hintergrund um unterschiedliche Dateiformate. Beides sind reine Textdateien (ASCII), allerdings lässt sich der Hintergrund direkt über dem MATLAB-Befehl `load` einlesen, wohingegen das mit den eigentlichen Absorptionsdaten nicht funktioniert, da hier noch zusätzliche Kopfzeilen („Header“) vorhanden sind und außerdem MATLAB als Dezimaltrennzeichen den Punkt und nicht das Komma erwartet[3].
- Zum Einlesen der Messdaten muss auf die grundlegenden Funktionen `fopen`, `fgetl` und `fclose` zurückgegriffen werden, um die Textdatei zeilenweise in ein `cell array` einzulesen.
- Da MATLAB `character arrays` nur mit identischen Dimensionen erlaubt[4], Sie aber nicht davon ausgehen können, dass jede Zeile Ihrer zu ladenden Textdatei die gleiche Länge hat, müssen Sie ein `cell array` zur Zwischenspeicherung des Dateiinhaltes verwenden.
- Die Definition der Variable `line` mit der Dimension 0 in Zeile 3 löst die ansonsten vom MATLAB-Editor (in Zeile 6) gezeigte Warnung, diese Variable ändere ihre Größe in einer Schleife. Strenggenommen tut sie das immer noch, aber das lässt sich wie oben ausgeführt hier auch gar nicht verhindern, zumal Ihnen die Länge der Datei im Vorfeld nicht bekannt ist.
- Die Konversion der eingelesenen Textdaten in numerische Daten, die mit MATLAB weiterverarbeitet werden können, erfolgt in einer weiteren Schleife (Zeilen 15–17), nachdem die Header-Zeilen abgeschnitten wurden (Zeile 13). Da jetzt die Dimension der Daten bekannt ist, kann die entsprechende Variable in der richtigen Größe vordefiniert werden (Zeile 14).
- Die Konversion von Text in numerische Daten erfolgt mehrstufig, wenngleich sehr komprimiert dargestellt, in Zeile 16. Ähnlich wie in der Mathematik ist es hilfreich, sich den Ablauf von innen nach außen anzuschauen. Hinzu kommt Hintergrundwissen über das Aussehen der Ausgangsdatei

[3] Das ist vermutlich der angelsächsischem Herkunft des Programmes geschuldet.
[4] Jede Zeichenkette („string“) muss die gleiche Länge haben.

(zwei Spalten mit Zahlen mit Komma als Dezimaltrennzeichen, getrennt durch Leerzeichen, die erste Spalte ist die Wellenlänge, die zweite die korrespondierende Intensität der Absorption).

Die Konversion in einzelnen Schritten:

a) Ersetzung des Dezimaltrennzeichens: `strrep(string,pattern,replacement)`
   Das Ergebnis ist wieder eine Zeichenkette (String).

b) Auftrennung der Zeile in zwei numerische Werte: `textscan(string,formatstring)`
   Das Ergebnis ist an dieser Stelle ein cell array mit zwei numerischen Feldern.
   Der Formatierungsstring ist sehr an die Gepflogenheiten der Programmiersprache C angelehnt. Details finden sich in der MATLAB-Hilfe. Das hier verwendete „`%f`“ steht für den numerischen Typ `float` (eine Gleitkommazahl).

c) Konversion des cell arrays in einen (numerischen) Vektor[5]: `cell2mat(cell)`
   Das Ergebnis dieser Operation ist ein Vektor mit zwei Elementen.

d) Zuweisung der Werte zur Matrix, die die Daten enthält: `data(k,:)=...`
   MATLAB indiziert immer in der Reihenfolge Reihe, Spalte. Hier steht die Laufvariable `k` also für die Zeile, der Doppelpunkt ist eine spezielle Notation für alle Elemente einer Dimension, hier die Spalte.

- Da das Hintergrundspektrum[6] in einem direkt von MATLAB einlesbaren Textformat vorliegt, kann hier der Befehl `load` verwendet werden.

- Der Hintergrund soll von den eigentlichen Messdaten abgezogen werden. Dazu müssen beide Spektren den gleichen spektralen Bereich abdecken. In unserem Fall wurde der Hintergrund über einen größeren Wellenlängenbereich aufgenommen und muss deshalb zurechtgeschnitten werden (Zeile 23).

- In einem nächsten Schritt wird der Hintergrund an seinem rechten spektralen Ende auf Null korrigiert (Zeile 24), da die ursprünglichen Daten zu längeren Wellenlängen hin gemessen wurden und deshalb die Autokorrektur des Spektrometers (*auto zero*) hier nach der Vorverarbeitung der Daten nicht mehr greift.

  Das Schlüsselwort `end` ist, ähnlich wie der Doppelpunkt, ein spezieller Index in MATLAB. Es referenziert immer auf den letzten Index einer gegebenen Dimension.

- Um den Hintergrund sinnvoll von den Messdaten abziehen zu können, muss er auf die Messdaten skaliert werden (Zeile 25). Hier wird die zusätzliche Information verwendet, dass ab 520 nm hin zu höheren Wellenlängen die eigentlich vermessene Spezies nicht mehr absorbiert, alle hier sichtbare Absorption also dem Streuhintergrund zuzuordnen ist.

  Hier sehen Sie eine weitere Form der (impliziten) Indizierung in MATLAB, die sehr mächtig ist und viel Schreibarbeit erspart: Um den Index des Vektors `data(:,1)` zu erhalten, dessen zugehöriger Wert 520 (entspricht der Wellenlänge) ist, kann man einfach `data(:,1)==520` schreiben.

- Nach all dieser Vorarbeit lässt sich der Hintergrund geradezu unspektakulär von den Messdaten abziehen (Zeile 26).

[5] Vektoren sind, genauso wie Matrizen, in MATLAB immer numerisch. Aus Sicht eines Programmierers handelt es sich bei den Vektoren und Matrizen von MATLAB um numerische Arrays.

[6] Hier handelt es sich um einen Streuhintergrund, der experimentell aufgenommen wurde. Gerade Proteinproben neigen dazu, einen solchen Streuhintergrund zu besitzen.

- Die grafische Darstellung der Messdaten ist eher unspektakulär und verwendet nur die normalen MATLAB-Befehle (`plot`, Zeile 29ff.).

  Da jeder einzelne `plot`-Befehl die Achsen löschte und neu zeichnete, was nicht erwünscht ist, wenn man zu einer bestehenden Achse zusätzliche Linien hinzufügen möchte, gibt es das Befehlspaar `hold on ... hold off`.

- Achsenbeschriftungen lassen sich einfach über die beiden Befehle `xlabel` und `ylabel` realisieren.

  Wie Sie erkennen können, verstehen diese Befehle grundlegende LaTeX-Formatierungsanweisungen. Achten Sie immer darauf, die Größen kursiv, die Einheiten aufrecht zu setzen und beide durch einen Schrägstrich zu trennen, der links und rechts von einem Leerzeichen umgeben ist.[7]

- MATLAB-Grafiken eignen sich in der Form, wie sie MATLAB erst einmal ausgibt, nicht für Publikationszwecke. Allerdings erlaubt MATLAB, fast alle Eigenschaften einer Abbildung zu kontrollieren.[8] Beispiele für die Kontrolle über das Aussehen einer MATLAB-Abbildung liefern die Zeilen 36ff. Sie werden sich noch etwas detaillierter in einer folgenden Lektion damit befassen.

- MATLAB-Abbildungen bestehen letztlich aus einer Reihe hierarchisch angeordneter Java-Objekte. Deshalb erfolgt die Abfrage und das Setzen von Eigenschaften (Attributen) über die in der objektorientierten Programmierung (OOP) üblichen Funktionen `get` und `set`. Details dazu wie schon angesprochen in einer späteren Lektion.

- Die Referenz auf ein Objekt wird in der MATLAB-Dokumentation `handle` genannt (dt. etwa *Griff*).

  Es gibt mehrere spezielle `handles` in MATLAB, von denen hier zwei Verwendung finden: `gca` ist immer eine Referenz auf die aktuell aktive Achse (*axis*), `gcf` auf die aktuell aktive Abbildung (*figure*).[9]

- Die grundlegende Syntax für die Befehle `get` und `set` lautet `get(handle,property)` bzw. `set(handle,property,value)` Die jeweiligen Eigenschaften und ihre möglichen Werte sind in der zugehörigen MATLAB-Hilfe ausführlich dokumentiert.

  Hier kann nicht für jede einzelne Codezeile im Detail beschrieben werden, was sie bewirkt. Versuchen Sie, eigenständig anhand der MATLAB-Hilfe zu verstehen, was genau passiert.

- Der Export der fertigen Abbildung als PDF-Datei ist erstaunlich einfach (Zeile 53). Konkret wird hier die Abbildung „gedruckt“ und dafür der PDF-Druckertreiber von MATLAB verwendet. Details dazu finden Sie in der MATLAB-Hilfe.

[7] Anmerkung: Die Angabe von „OD“ als Einheit für die Absorption ist bestenfalls umstritten, schlimmstenfalls falsch. Siehe dazu u.a. auch: Kamat, P. und G. C. Schatz (2013): How to make your next paper scientifically effective, *J. Phys. Chem. Lett.* **4**:1578–1581.

[8] Inwieweit das erreichbare Ergebnis tatsächlich publikabel ist, hängt nicht zuletzt von den eigenen Ansprüchen ab. Ein entscheidender Vorteil einer Verarbeitung der Abbildungen direkt mit MATLAB ist, dass bei Änderungen in der Datenverarbeitung nicht jedesmal Abbildungen wieder per Hand (z.B. mit dem Adobe Illustrator oder vergleichbaren Programmen) nachbearbeitet werden müssen.

[9] `gca` steht für *get current axis*, `gcf` respektive für *get current figure*. Beide liefern das entsprechende `handle` zurück.

Listing 2: Dokumentierter und in Blöcke aufgeteilter Quellcode

```
% Load data file
fid=fopen('uvvis.txt');
line=cell(0);
k=1;
while ~feof(fid)
    line{k}=fgetl(fid);
    k=k+1;
end
fclose(fid);

% Convert data
% Ignore header lines
line(1:2)=[];
data=zeros(length(line),2);
for k=1:length(line)
    data(k,:)=cell2mat(textscan(strrep(line{k},',','.'),'%f %f'));
end

% Load background
bg=load('bg.txt');

% Cut and scale background, subtract background from data
bg=bg(1:length(data),:);
bg(:,2)=bg(:,2)-bg(end,2);
bg(:,2)=bg(:,2)*(data(data(:,1)==520,2)/bg(bg(:,1)==520,2));
data(:,2)=data(:,2)-bg(:,2);

% Plot data
plot(data(:,1),data(:,2),'k-');
hold on;
plot(data(:,1),zeros(length(data(:,2)),1),'k--');
hold off;
xlabel('{\it wavelength} / nm');
ylabel('{\it intensity} / OD');

% Set fonts and paper/figure sizes
set(get(gca,'xlabel'),'fontsize',12);
set(get(gca,'ylabel'),'fontsize',12);
set(get(gca,'xlabel'),'fontname','Arial');
set(get(gca,'ylabel'),'fontname','Arial');
set(gca,'fontsize',12);
set(gca,'fontname','Arial');
set(gcf,'paperunits','centimeters');
set(gcf,'papersize',[16 10]);
set(gcf,'paperpositionmode','auto');
set(gca,'Units','centimeters');
set(gca,'OuterPosition',[0 0 16 10]);
set(gcf,'Units','centimeters');
oldpos = get(gcf,'Position');
set(gcf,'Position',[oldpos([1 2]) 16 10]);

% Export figure as PDF
print(gcf,'data.pdf','-dpdf');
```

Aufgabe 4—3 (Quellcode in Funktionen aufteilen)

War die erste Aufgabe, das reine Formatieren des Quellcodes, noch in weiten Teilen deterministisch und das Ergebnis eindeutig, ermöglicht schon die Unterteilung des Skriptes in einzelne Blöcke inklusive kurzer Dokumentation eine erhebliche Freiheit des einzelnen Nutzers. Das gilt erst recht für die Unterteilung in einzelne Funktionen, deren Benennung, deren Schnittstellendesign und Dokumentation. Deshalb sei noch einmal betont: Das nachfolgend dokumentierte Beispiel ist in keiner Weise allgemeingültig und mit hoher Sicherheit weit davon entfernt, die ideale Lösung darzustellen. Des Weiteren sei angemerkt, dass eine praxistaugliche Lösung einiger weiterer Verallgemeinerungen bedürfte, die hier aus didaktischen Gründen erwähnt, aber nicht umgesetzt werden sollen.

Grundsätzlich eignet sich als Vorgehen für eine Modularisierung und Aufteilung in einzelne Funktionen die im Rahmen der Bearbeitung der vorangegangenen Aufgabe erfolgte Unterteilung in funktionale Blöcke. Es bietet sich an, hier das „Unix-Prinzip“ anzuwenden: Eine Aufgabe, eine Routine. Ein kurzes Skript dient dann dem Aufruf der einzelnen Funktionen.

Schon die Zahl der (zu schreibenden) Funktionen ist eine Frage des persönlichen Geschmacks. Das in Listing 3 wiedergegebene Beispiel nutzt vier Funktionen:

a) Daten einlesen

b) Hintergrund abziehen

c) Daten darstellen

d) Abbildung speichern

Klar erkennbar ist hier die Verwendung sprechender Funktionsnamen, die dazu führen, dass das Skript auch ohne weitere Kommentare lesbar und verständlich ist.

Listing 3: Skript mit Funktionsaufrufen zur Datenverarbeitung

```
% Load UV/vis data and background, subtract background, plot and export.

% Copyright (c) 2014-16, Till Biskup <till.biskup@physchem.uni-freiburg.de>
% 2016-07-31

data = loadUVVisData('uvvis.txt');
background = loadUVVisData('bg.txt');
data = subtractBackgroundFromData(background,data);
plotUVVisData(data,'zeroLine',true);
figure2file(gcf,'data.pdf');
```

Tatsächlich ist es manchmal hilfreich, sich den grundlegenden Ablauf zu überlegen, entsprechende Funktionsaufrufe in einem Skript (wie in Listing 3 gezeigt) untereinander zu schreiben und dann erst damit zu beginnen, die einzelnen Funktionen zu implementieren. Ggf. ergeben sich in diesem Prozess noch Änderungen an den Schnittstellen oder dem Zuschnitt einzelner Funktionen.

Nachfolgend werden für jede der vier hier vorgeschlagenen Funktionen beispielhafte Implementationen vorgestellt. Dabei handelt es sich keinesfalls um Referenzimplementationen, sondern um Beispiele zur Illustration. Für einen produktiven Einsatz wären noch weitere Modularisierungen und Erweiterungen hin zu robusterem Code unerlässlich.

Listing 4: Funktion zum Einlesen von UV/vis-Daten

```
1  function data = loadUVVisData(filename)
2  % LOADUVVISDATA Import UV/vis data from Shimadzu ASCII export.
3  %
4  % Usage
5  %   data = loadUVVisData(filename)
6  %
7  %   filename - string
8  %              name of the file to load
9  %
10 %   data     - matrix (nx2)
11 %              absorption data
12 %              1st column: wavelengths
13 %              2nd column: intensities
14 %
15 % Note: Bare ASCII data files consisting only of two columns representing
16 % wavelength and intensity can be read as well.
17
18 % Copyright (c) 2014-16, Till Biskup <till.biskup@physchem.uni-freiburg.de>
19 % 2016-07-31
20
21 % Number of header lines
22 nHeaderLines = 2;
23
24 % Try first with load command,
25 % and if that fails, assume Shimadzu ASCII export format.
26 try
27     data = load(filename);
28     return;
29 catch %#ok<CTCH>
30     fid=fopen(filename);
31     line=cell(0);
32     k=1;
33     while ~feof(fid)
34         line{k}=fgetl(fid);
35         k=k+1;
36     end
37     fclose(fid);
38 end
39
40 % Ignore header lines
41 line(1:nHeaderLines)=[];
42
43 % Convert data
44 data=zeros(length(line),2);
45 for k=1:length(line)
46     data(k,:)=cell2mat(textscan(strrep(line{k},',','.'),'%f %f'));
47 end
```

Anmerkungen:

- Konstanten[10] wie die Zahl der Zeilen des Dateikopfes werden am Beginn der Funktion definiert.
- Die Funktion versucht zunächst, die Daten mittels *load*-Befehl zu laden (Zeile 27). Nur im Misserfolgsfall werden die Daten manuell gelesen und konvertiert.

[10] MATLAB unterscheidet in unserem Kontext nicht zwischen Konstanten und Variablen.

Listing 5: Funktion zur Hintergrundkorrektur

```
1  function correctedData = subtractBackgroundFromData(background,data)
2  % SUBTRACTBACKGROUNDFROMDATA Subtract background from absorption data.
3  %
4  % Usage
5  %   data = subtractBackgroundFromData(background,data);
6  %
7  %   background    - matrix (nx2)
8  %                   background data
9  %                   1st column: wavelengths
10 %                   2nd column: intensities
11 %
12 %   data          - matrix (nx2)
13 %                   absorption data
14 %                   1st column: wavelengths
15 %                   2nd column: intensities
16 %
17 %   correctedData - matrix (nx2)
18 %                   corrected absorption data
19 %                   1st column: wavelengths
20 %                   2nd column: intensities
21
22 % Copyright (c) 2014-16, Till Biskup <till.biskup@physchem.uni-freiburg.de>
23 % 2016-07-31
24
25 % Wavelength to scale background to data at (in nm)
26 scalingWavelength = 520;
27
28 % Assign output
29 correctedData=data;
30
31 % Cut background
32 background=background(1:length(data),:);
33
34 % Auto zero background
35 background(:,2)=background(:,2)-background(end,2);
36
37 % Scale background
38 background(:,2)=background(:,2)*(data(data(:,1)==scalingWavelength,2)...
39     /background(background(:,1)==scalingWavelength,2));
40
41 % Subtract background from data
42 correctedData(:,2)=data(:,2)-background(:,2);
```

Anmerkungen:

- Die Funktion ist in ihrer hier vorgestellten Form wenig flexibel, u.a. wegen der Festlegung der Wellenlänge, bei der der Hintergrund auf die Daten skaliert wird (Zeile 26).
- Das Zurechtschneiden des Hintergrundes geht davon aus, dass sowohl die kürzeste Wellenlänge als auch die Schrittweite der Wellenlängenachse für Hintergrund und Daten identisch sind. Das sind in der Praxis mehr als optimistische Annahmen.

  Zur Verbesserung kann u.a. logische Indexierung (vgl. Anmerkungen zu Listing 2) eingesetzt werden.
- Bei unterschiedlichen Achsen kann interpoliert werden (siehe die Hilfe zu `interp1`).

Listing 6: Funktion zum Darstellen von UV/vis-Daten

```
function plotUVVisData(data,varargin)
% PLOTUVVISDATA Plot UV/vis data.
%
% Usage
%   plotUVVisData(data);
%   plotUVVisData(data,'zeroLine',true);
%
%   data          - matrix (nx2)
%                   absorption data
%                   1st column: wavelengths
%                   2nd column: intensities
%
% Optional parameters (key-value pairs)
%
%   zeroLine      - boolean (OPTIONAL)
%                   whether to plot a dashed zero line

% Copyright (c) 2014-16, Till Biskup <till.biskup@physchem.uni-freiburg.de>
% 2016-07-31

% Parse input arguments using the inputParser functionality.
p = inputParser;             % Create an instance of the inputParser class.
p.FunctionName  = mfilename; % Include function name in error messages.
p.KeepUnmatched = true;      % Enable errors on unmatched arguments.
p.StructExpand  = true;      % Enable passing arguments in a structure.

p.addRequired('data',@(x)isnumeric(x) && ismatrix(x) && size(x,2)==2);
p.addParamValue('zeroLine',logical(false),@islogical);
p.parse(data,varargin{:});

% Plot data
plot(data(:,1),data(:,2),'k-');

% Plot zero line
if p.Results.zeroLine
    hold on;
    plot(data(:,1),zeros(length(data(:,2)),1),'k--');
    hold off;
end

xlabel('{\it wavelength} / nm');
ylabel('{\it intensity} / OD');
```

Anmerkungen:

- Die Funktion versteht zusätzliche (optionale) Parameter (Übergabeparameter `varargin`).

  Das Design der Schnittstelle (vgl. Listing 3) sieht Schlüssel-Wert-Paare vor.

- Zur Handhabung zusätzlicher Parameter, insbesondere in ihrer hier verwendeten Form als Schlüssel-Wert-Paare, wird die Klasse `inputParser` verwendet. Details dazu werden im Rahmen dieses Kurses nicht besprochen, finden sich aber in der MATLAB-Dokumentation (`doc inputParser`).

Listing 7: Funktion zum Export von MATLAB-Abbildungen

```
1  function figure2file(figureHandle,filename)
2  % FIGURE2FILE Export Matlab figure to (PDF) file.
3  %
4  % Usage
5  %   figure2file(figureHandle,filename)
6  %
7  %   figureHandle - graphics handle
8  %                  valid Matlab figure handle
9  %
10 %   filename     - string
11 %                  filename to save the figure to
12
13 % Copyright (c) 2014-16, Till Biskup <till.biskup@physchem.uni-freiburg.de>
14 % 2014-07-31
15
16 % User settings
17 fontSize = 12;
18 fontName = 'Arial';
19 figureUnits = 'centimeters';
20 figureDimensions = [16 10];
21
22 % Get axis handle of figureHandle
23 axisHandle = findobj('Parent',figureHandle,'Type','axes');
24
25 % Set fonts
26 set([get(axisHandle,'xlabel') get(axisHandle,'ylabel') axisHandle],...
27     'fontsize',fontSize,...
28     'fontname',fontName ...
29     );
30
31 % Set paper/figure sizes
32 set(axisHandle,...
33     'Units',figureUnits,...
34     'OuterPosition',[0 0 figureDimensions] ...
35     );
36 oldPosition = get(figureHandle,'Position');
37 set(figureHandle,...
38     'paperunits',figureUnits,...
39     'papersize',figureDimensions,...
40     'paperpositionmode','auto',...
41     'Units',figureUnits,...
42     'Position',[oldPosition([1 2]) figureDimensions] ...
43     );
44
45 % Export figure as PDF
46 print(figureHandle,filename,'-dpdf');
```

Anmerkungen:

- Die nutzerspezifischen Einstellungen stehen am Anfang der Funktion (Zeilen 17–20).
- Mehrere „Handles“ können in einem `set`-Befehl zusammengefasst werden (Zeile 26).
- Mehrere Eigenschaften können mit einem `set`-Befehl gesetzt werden.

Einige weitergehende Anmerkungen zu den hier vorgestellten Funktionen:

- Alle Funktionen und das sie aufrufende Skript (Listing 3) wurden in der hier abgebildeten Form in MATLAB (Version 2014a) getestet.
- Jede Funktion hat am Anfang einen Kommentarkopf, der dem in der Präsentation vorgestellten Schema folgt. Er enthält sowohl eine kurze Beschreibung der Funktion als auch ihrer Schnittstelle. Gefolgt wird dieser Kommentarkopf vom Urheberrechtshinweis mit Datum der letzten Änderung.

  Achten Sie darauf, bei Ihren eigenen Dateien dieses Änderungsdatum immer zu pflegen. Das Änderungsdatum einer Datei auf dem Dateisystem ist nicht immer zuverlässig (z.B., wenn die Datei verschoben oder kopiert wurde).
- Die hier vorgestellten Funktionen sind nur *eine Möglichkeit* der Umsetzung.

  Für den arbeitstäglichen praktischen Einsatz sollten sie mitunter noch etwas weiter modularisiert und robuster programmiert werden.
- Jede Funktion versucht, genau eine Aufgabe zu erfüllen.

  Die Funktionen sind kurz und übersichtlich gehalten. Einige Funktionen (insbesondere zur Hintergrundkorrektur, `subtractBackgroundFromData`) setzen noch zu viel an Spezialwissen bzw. speziellen Gegebenheiten voraus.
- Die Aufteilung der Datenverarbeitung in einzelne Funktionen (Listung 3) war eine logische Folge der Untergliederung des ursprünglichen Quellcodes (Listing 1) in einzelne Abschnitte (Listing 2).

  Das Design der Schnittstellen entstand für das vorliegende Beispiel „am grünen Tisch“, allerdings gegründet auf Hintergrundwissen und langjährige Erfahrung. Die Funktionen wurden danach entsprechend dieses Schnittstellenentwurfs umgesetzt und folgen in weiten Teilen dem ursprünglichen Code.
- Einige Details in den Funktionen wurden gegenüber dem ursprünglichen Quellcode abgeändert.

  Insbesondere Variablennamen und Layout des Quellcodes wurden angepasst, um die Verständlichkeit und Lesbarkeit zu erhöhen.

Trotz aller Widrigkeiten und Vereinfachungen sollte an diesem Beispiel deutlich geworden sein, wie sich die Konzepte Modularität und Dokumentation im Code (insbesondere Schnittstellendokumentation) in der Praxis umsetzen lassen.

Abschliessend seien Ihnen einfach zu merkende Akronyme zweier wesentlicher Aspekte modularen Programmierens genannt:

DRY *Don't repeat yourself*

KISS *Keep it simple and stupid*

Vergessen Sie nicht, jede auch noch so einfache Funktion zumindest grundlegend zu dokumentieren. Ein Einzeiler zur Funktionsbeschreibung, eine Dokumentation der Schnittstelle und der Urheberrechtshinweis mit (gepflegtem und richtigem) Datum der letzten Änderung sind obligatorisch.

Aufgabe 4—4 (Bisher geschriebene Funktionen dokumentieren)

Nach dieser vergleichsweise komplexen Aufgabe, ein Skript in einen Satz von Funktionen zu zerlegen, die jede eine Aufgabe erfüllen, und dann diese Funktionen auch noch sauber zu dokumentieren, sollte es relativ einfach sein, das neu erarbeitete Wissen auf die im Rahmen der Bearbeitung vorangegangener Aufgabenblätter erstellten Funktionen `gaussian` und `rotationMatrix` anzuwenden.

Beide Funktionen erfüllen dankenswerter Weise bereits ein wesentliches Kriterium von Funktionen: Sie führen jeweils genau eine Aufgabe aus, es gibt also keinen Grund, sie weiter in Unterfunktionen zu zerlegen.

Nachfolgend zunächst die kommentierte und dokumentierte Fassung der Funktion `gaussian` zur bequemen Erzeugung von Gauß-Kurven für einen gegebenen Vektor $\boldsymbol{x}$.

Listing 8: Kommentierte und dokumentierte Fassung der Funktion `gaussian.m`

```
function y = gaussian(x,sigma,mu)
% GAUSSIAN Create Gaussian curve with width sigma and position mu.
%
% Usage
%   y = gaussian(x,sigma,mu)
%
%   x     - vector
%           x values to evaluate the Gaussian for
%
%   sigma - scalar
%           width of the Gaussian
%
%   mu    - scalar
%           position of maximum of the Gaussian
%
%   y     - vector
%           resulting Gaussian evaluated at positions x

% Copyright (c) 2016, Till Biskup <till.biskup@physchem.uni-freiburg.de>
% 2016-07-31

y = 1/sqrt(2*pi*sigma^2).*exp(-(x-mu).^2./(2*sigma^2));

end
```

Diese Funktion ist ein gutes Beispiel dafür, dass ein großer Teil des gesamten Dateiinhaltes für die Dokumentation verwendet wird. Der eigentlich produktive Quellcode beschränkt sich auf eine einzelne Zeile.

Zum einen wird hier einmal mehr deutlich, wie elegant sich auch relativ komplexe mathematische Funktionen in einer Hochsprache wie MATLAB intuitiv implementieren lassen. Zum anderen sollten Sie die Bedeutung des (im Verhältnis zum produktiven Quellcode hier viel größeren) Kommentarkopfes nicht unterschätzen. Er wird als Hilfe über die Befehle `help` und `doc` ausgegeben und liefert aufgrund seiner Strukturierung dem Nutzer einen schnellen Überblick über Aufgabe und Aufruf der Funktion.

Abschließend nun noch eine kommentierte und dokumentierte Version der Funktion `rotationMatrix` zur Erzeugung von Euler-Drehmatrizen aus drei übergebenen Euler-Winkeln. Hier darf in der Dokumentation der Funktion der Hinweis nicht fehlen, um welche Konvention es sich handelt, da das ein Detail ist, das aus dem Funktionsnamen nicht hervorgeht. Da man sich normalerweise aber in einem gegebenen Kontext auf eine Konvention beschränkt, ist das kein Problem. Sollten mehrere Funktionen für

mehrere unterschiedliche Konventionen nebeneinander existieren, sollte sich die jeweilige Konvention aber selbstverständlich im Funktionsnamen widerspiegeln.

Listing 9: Kommentierte und dokumentierte Fassung der Funktion `rotationMatrix.m`

```
function M = rotationMatrix(alpha,beta,gamma)
% ROTATIONMATRIX Create Euler rotation matrix from set of Euler angles.
%
% The rotation follows the zy'z'' convention.
%
% Usage
%   M = rotationMatrix(alpha,beta,gamma)
%
%   alpha - scalar
%           Euler angle for first rotation
%
%   beta  - scalar
%           Euler angle for second rotation
%
%   gamma - scalar
%           Euler angle for third rotation
%
%   M     - 3x3 matrix
%           Euler rotation matrix for the given set of Euler angles

% Copyright (c) 2016, Till Biskup <till.biskup@physchem.uni-freiburg.de>
% 2016-07-31

M = [...
    -sin(alpha)*sin(gamma)+cos(alpha)*cos(beta)*cos(gamma) ...
    cos(alpha)*sin(gamma)+sin(alpha)*cos(beta)*cos(gamma) ...
    -sin(beta)*cos(gamma) ; ...
    %
    -sin(alpha)*cos(gamma)-cos(alpha)*cos(beta)*sin(gamma) ...
    cos(alpha)*cos(gamma)-sin(alpha)*cos(beta)*sin(gamma) ...
    sin(beta)*sin(gamma) ; ...
    %
    cos(alpha)*sin(beta) ...
    sin(alpha)*sin(beta) ...
    cos(beta) ...
    ];

end
```